IN THE CLAIMS:

The text of all pending claims, (including withdrawn claims) is set forth below. Cancelled and not entered claims are indicated with claim number and status only. The claims as listed below show added text with underlining and deleted text with ~~strikethrough~~. The status of each claim is indicated with one of (original), (currently amended), (cancelled), (withdrawn), (new), (previously presented), or (not entered).

Please ADD new claim 26 in accordance with the following:

1. (PREVIOUSLY PRESENTED) A method, comprising:

evaluating a dependency graph of a graphics creation process using a computer, comprising:

passing a pointer to an algorithm associated with a first dependency node to a second dependency node allowing the second dependency node to execute the algorithm; and

executing the algorithm as part of an evaluation of the second dependency node.

2. (PREVIOUSLY PRESENTED) A method as recited in claim 1, wherein the algorithm comprises a self evaluating data structure.

3. (PREVIOUSLY PRESENTED) A method as recited in claim 2, wherein the algorithm comprises an algorithm having a defined set and type of inputs and outputs.

E1

4. (CANCELLED)

5. (PREVIOUSLY PRESENTED) A method as recited in claim 2, wherein the structure comprises an algorithm calling method.

6. (ORIGINAL) A method as recited in claim 2, wherein the evaluating comprises determining a type of a passed parameter.

7. (PREVIOUSLY PRESENTED) A method as recited in claim 6, wherein the algorithm parameter types are identified dynamically as the dependency graph is executed.

8. (PREVIOUSLY PRESENTED) A method as recited in claim 7, wherein the data structure contains information describing a set of input and output parameters the algorithm

accepts.

9. (PREVIOUSLY PRESENTED) A method as recited in claim 8, wherein the information determines if algorithm attribute types within the dependency graph are compatible.

10. (ORIGINAL) A method as recited in claim 9, wherein the data structure comprises default values for all input and output parameters.

11. (PREVIOUSLY PRESENTED) A method as recited in claim 1, further comprising mapping parameters of first and second algorithms of the first and second nodes.

12. (ORIGINAL) A method as recited in claim 11, wherein said mapping comprises using an index.

13. (ORIGINAL) A method as recited in claim 11, wherein the mapping defines a relationship where input parameters are ignored and output parameters are unmapped and take on default values.

14. (ORIGINAL) A method as recited in claim 11, wherein parameter value and type are passed for the mapping.

15. (PREVIOUSLY PRESENTED) A method as recited in claim 11, wherein the algorithm data structure and value index are passed for the mapping.

16. (ORIGINAL) A method as recited in claim 11, wherein the mapping comprises an index remapping and a matrix of data casting methods which will change one type of data into another.

17. (PREVIOUSLY PRESENTED) A method, comprising:

evaluating a dependency graph of a graphics creation process using a computer, comprising:

passing a pointer to an algorithm of a first dependency node to a second dependency node allowing the second dependency node to execute the algorithm of the dependency node, the algorithm comprising a self evaluating data structure comprising an algorithm calling method

and containing information describing a set of input and output parameters the algorithm accepts where the information determines if algorithm attribute types within the dependency graph are compatible and comprising default values for all input and output parameters;

mapping parameters of first and second algorithms of the first and second nodes, where the mapping comprises an index, defines a relationship where input parameters are ignored and output parameters are unmapped and take on default values, where parameter value and type are passed for the mapping and the algorithm data structure and value index are passed for the mapping; and

executing the algorithm of the first dependency node as part of an evaluation of the second dependency node using the pointer, each time input data changes, and determining a type of a passed parameter where parameter types are identified dynamically as the dependency graph is executed.

18. (ORIGINAL) A method as recited in claim 17, wherein the mapping comprises an index remapping and a matrix of data casting methods which will change one type of data into another.

19. (PREVIOUSLY PRESENTED) A method, comprising:

evaluating a dependency graph of a graphics creation process using a computer, comprising:

passing a pointer to an algorithm from a first node in a node network to a second node in the node network allowing the second node to execute the algorithm; and

executing the algorithm as part of an evaluation of the second node each time input data changes.

20. (PREVIOUSLY PRESENTED) An apparatus comprising a computer including a dependency node evaluation system having pointers to algorithms passed between nodes of a dependency graph of a graphics creating process allowing the nodes receiving the pointers to execute the algorithms, the system evaluating the dependency graph based on the algorithms passed between the nodes.

21. (PREVIOUSLY PRESENTED) A data structure provided on computer readable storage controlling a computer in association for evaluating a dependency graph of a graphics creation process, the data structure comprising a run time type identification parameter list, a

mapping substructure comprising an index mapping, mapping methods, and a data casting matrix, an algorithm pointer, and methods for setting inputs, getting outputs, and evaluating an algorithm using the passed pointer.

22. (PREVIOUSLY PRESENTED) A method, comprising:

evaluating a dependency graph of a graphics creation process using a computer, comprising:

performing, by a destination node, of an algorithm having a function known to the destination node by evaluating a self evaluating data structure passed from a source node to the destination node using a pointer to the algorithm

allowing the destination node to execute the algorithm of the source node and the structure expected to precisely implement the algorithm known to the destination node where the self evaluating data structure can comprise a different algorithm with different parameters and performing the different algorithm actually requested by the destination node.

23. (PREVIOUSLY PRESENTED) A method of evaluating a dependency graph, comprising:

passing an algorithm associated with a first dependency node to a second dependency node;

executing the algorithm via the second node as part of an evaluation of the second node; and

reexecuting the algorithm via the second node each time input data of the second node changes.

24. (PREVIOUSLY PRESENTED) A method of evaluating a dependency graph, comprising:

passing an algorithm associated with a first dependency node to a second dependency node; and

executing the algorithm via the second node as part of an evaluation of the second node.

25. (PREVIOUSLY PRESENTED) A method of evaluating a dependency graph of a graphics creation process using a computer, comprising:

passing a pointer to an algorithm associated with a first dependency node to a second dependency node allowing the second dependency node to execute the algorithm; and

calling the algorithm via the second dependency node, and executing the algorithm as part of an evaluation of the second dependency node each time input data changes.

61 26. (NEW) A method of evaluating a dependency graph of a graphics creation process using a computer, comprising:

passing a first algorithm associated with a first dependency node to a second dependency node allowing the second dependency node having a second algorithm to execute the first algorithm, wherein the first algorithm is embedded in the second dependency node and executed as part of an evaluation of the second dependency node.